Summary on
CLIO: Enabling automatic compilation of deep learning pipelines across IoT and Cloud

Lohitanvita Rompicharla

RUID: 211009876

Department of Electrical and Computer Engineering

Objective

In the world of automation, there is always a constant re-invention for the improvement of efficiency, versatility, and adaptability. There have been many advancements in technology recently, such as fusion of IoT and sensor networks, low-power neural accelerators, low-power radios, Neuro.Zero, using data compression, model compression, or hand-crafted techniques for incorporating deep learning models into embedded devices. All these methods have achieved their goal of using machine learning and deep learning models into IoT devices but still are subject to some limitations like bandwidth, energy, computation ability, reduction in inference performance, etc. Taking these limitations into consideration, CLIO (**Cl**oud – **Io**T partitioning), an end-to-end approach is implemented that practices model compilation for low-power IoT devices by automatically splitting deep learning model between IoT device and cloud, optimizing latency, energy, and overall inference performance.

Introduction

Latest inventions involving IoT and sensors like thermostats, cameras, microphones, IMU, and smartphones are efficiently supporting continuous analytics with low power consumption. But they are still limited to resource constraints for efficient execution of state-of-art deep learning models into low-power IoT devices. The present techniques for integrating DNN models are data compression where a large data is squeezed by transmitting a lossy version of data for remote execution, model compression where larger models are squeezed for local execution and, model partitioning which is a combination of both local and remote computing. These techniques are successful enough for incorporating deep learning models into embedded devices like smartphones, Raspberry PI, digital watches, etc., but have proved inefficient in the case of low-power accelerators and low-power radios. For low-power accelerators like GAP8 (1 GFlops) which have the basic architecture for the execution of deep learning models are 2-3 orders less in magnitude of resources compared to embedded devices, making model compression technique inefficient to use because of its restrictions in computation ability, memory size, maximum operating frequency. So there has been a replacement for model compression with hand-crafted models that are suitable for resource-constrained MCUs, ex: CMSIS-NN for ARM (80% accuracy), MobileNewV2 for CIFAR-10 (93% accuracy) but with notable performance degradation. For low-power radio like BLE, Zigbee, with high duty cycle efficiency, data compression method is helpful, but its uplink bandwidth is prone to many factors like body blockage, device mobility, attenuation due to walls, environmental dynamics. Taking all these factors into account, a model compilation framework, CLIO is introduced that executes deep learning models across IoT-Cloud while respecting resource-constraints and adapting to wireless variability. CLIO can compile state-of-art DL models like MobileNetV2, ResNet-18 by integrating progressive transmission with model compression and adaptive partitioning in response to wireless dynamics even in resource-constrained devices by achieving an end-to-end reduction in latency, power consumption, and inference accuracy.

Working

The Cloud-IoT partitioning (CLIO) approach works on the principle of progressive slicing and is adaptive to bandwidth dynamics. For example, considering a model M with 5 layers (L), input data D = xn, computation graph implementing function Ok = fΘ(x), slicing function O1:i = slice(i, O), and final predicted function y = gΦ (slice (i, fΘ(x))). IoT device executes the initial 3 layers, and the remaining 2 layers in the pipeline are executed at the cloud, then the output of layer 3 is partitioned into slices by slicing function depending on the bandwidth and transmits these slices to cloud by adding loss function l(y, y'). During inference, IoT device passes layers 1-3 to cloud via radio transmission and the radio sends all the data available to the cloud concerning network dynamics. In this process of progressive slicing, the model learns ordered hidden representation to minimize expected error by the addition of a lossy function that enables to predict expected output with distribution over the available bandwidth. The above example is used to train CLIO for IoT side and model compression for adaptive partition is used for training cloud side model layers. CLIO constantly adapts to the size of intermediate activations to achieve the best inference under different bandwidth conditions. Hence, the sequential procedure for CLIO training involves pruning(Meta-pruning, AutoML) before progressive slicing and weight quantization, sparsification after progressive slicing. The mentioned training methods are efficient in predicting error-free outputs but may induce some extra training time due to number of slices and number of partitioning points. Therefore, to reduce training complexity across slicing points number of different hidden sizes are limited and parameters starting at layer K+2 are tied so that only first layer parameters are expanded, greedy algorithm is used to choose potential partition points in the bandwidth tolerance range of progressive slicing. These implementations in CLIO help to accelerate and optimize the training time.

Compilation

The hardware setup of Clio contains an Integrated GAP8 node, HIMAX HM01B0 camera, BLE shield, and LCD screen. Clio compiles several different state-of-art neural network models like VGG, MobileNetV2, ResNet into two sub-models, that is locally executed on IoT devices and remotely executed in cloud servers using IoT processors namely GAP8 with Hardware Convolution Engine to accelerate convolutional operations, STM32F, an ARM7 based processor. Here the HIMAX camera captures images that are displayed on the LCD screen, then GAP8 integrated board executes the first layers of the model with the help of PyTorch/TFLite software and transmits the intermediate results to cloud via radio. In the cloud several benchmarks like Xeon Silver, GTX 1080 Ti GPU are used to predict the output. Then with the help of obtained output results of different computational blocks, we can conclude that the estimated latency values of GAP8 when executing MobileNetV2 is 525 MMAC/s, and transfer speed from GAP8 to BLE shield is 250kbps

Evaluation

For evaluation of Clio under numerous circumstances, CIFAR-10 and ImageNet datasets are utilized in separate workload methods. The outcome received from these datasets are evaluated and compared with various models and compression techniques such as DeepN-JPEG, Baseline JPEG, meta-pruning, width-multiplier, AutoML and determined that Clio performs well at latency above 220ms, since it gets sufficient latency for compute and data transfer for both local and remote processing. To evaluate partition point adaptability, synthetic Trace-driven Evaluation, and Real-world Trace-driven evaluation are employed where Clio is tested and compared by considering fixed partitioning and by varying amount of dynamics. Analyzing all the comparisons, it can be stated that progressive slicing, adaptive partitioning are better than model compression and progressive JPEG with an increase in accuracy.

Conclusion

Scrutinizing all the advantages, disadvantages, and compatibility of Clio with various other models like JALAD, Neurosurgeon, 2-step pruning we can conclude that CLIO provides better accuracy even in low compression ratios. It overcomes inference performance, bandwidth, resource-constraint issues by supporting an end-to-end optimized resolution. It also overrides the usage of Neuro.Zero by utilizing low-power devices.

Future scope

Based on Clio’s performance for image prediction deep learning models, this work can be extended to audio processing models in noisy environments, video processing models under various dependencies like LSTMS, 3D convolutions, etc. as Clio can automatically compile deep learning pipelines across edge and cloud.